Windows

OS-9 Windowing System

Contents

Chapter 1

Types of OS-9 Windows

Unlike many operating systems, OS-9 has a built-in windows program. This driver, the Windowing System, lets you lay one or more smaller screen displays, called *windows*, on your screen display.

With these windows, you can perform several tasks at the same time. For example, suppose you are writing a business letter using a word processor in one window. You can go to a spreadsheet program in another window, get a price quote you need, return to the word processor, and include the price in the letter.

The Windowing System allows as many windows as your computer's memory can support, with a maximum of 32 at one time.

In OS-9, there are two types of windows: device and overlay.

Device Windows

A *device window* is one that can run a program or utility. This is the type of window you would use in the word processor/spreadsheet example given above. Each device window acts as an individual terminal.

The device windows are designated as devices **/w1** - **/w7**. You open a device window as you do any other OS-9 device. You tell OS-9 the window's parameters—including whether the window is for text or graphics. If you want to run a process in the window, you can start an execution environment, such as a *shell*, on the window. (See "Opening a Device Window," later in this chapter, and the DWSet command in Chapter 3.)

> **Note:** If you want only to send output to the device window—without running a process in the window—do not start a shell on the window.

Device windows cannot overlay each other, and their boundaries cannot overlap.

Overlay Windows

An *overlay window* is a window that you open on top of a device window. (You can place overlay windows over other overlay windows, but there must always be a device window at the bottom of the stack.) The purpose of overlay windows is to display computer dialog. You cannot fork a shell to an overlay window; however, you can **run** a shell in an overlay window. Overlay windows assume the screen type of the device windows they overlay.

Opening a Device Window

To open a device window, follow these steps:

1. If you want to allocate memory for the window, use OS-9's **iniz** command. Type:

   ```
   iniz /wnumber [ENTER]
   ```

   where *number* is the number of the device window you wish to open (1-7). If you do not specify *number*, OS-9 uses the next available device window number.

   If you do not use the **iniz** command, memory is allocated dynamically (as needed) to the window.

2. Next, you send an escape sequence to OS-9 that tells it the window's parameters. These parameters include the screen type, size, and colors. For example:

   ```
   wcreate /w -s=2 20 10 40 10 01 00 00 [ENTER]
   ```

   or

   ```
   display 1b 20 02 14 0a 28 0a 01 00 00 /w
   [ENTER]
   ```

   sends the escape sequence for the next available window to the DWSet command. The **wcreate** command lets you use decimal numbers, while the **display** command requires hexadecimal numbers.

If you wish to send an escape sequence to a specific window, route the command to that device. For example:

```
wcreate /w2 -s=2 20 10 40 10 01 00 00 [ENTER]
```

sends the escape sequence to device **/w2**. The functions of the codes, as used in the **wcreate** command, are as follows:

2	sets a screen type of 80 x 24 (text only).
20	starts the window at character/column 20.
10	starts the window at line/row 10.
40	sets a window size of 40 characters.
10	sets a window size of 10 lines.
01	sets the foreground color to blue.
00	sets the background color to white.
00	sets the border to white.

If you do not send escape sequences, OS-9 uses default descriptors for the windows. The defaults are:

Window Number	**Screen Type (chars./line)**	**Starting Position (horiz., vert.)**	**Size (columns, rows)**
1	40 (text)	0,0	27,11
2	40 (text)	28,0	12,11
3	40 (text)	0,12	40,12
4	80 (text)	0,0	60,11
5	80 (text)	60,0	19,11
6	80 (text)	80,0	80,12
7	80 (text)	0,0	80,24

3. Use OS-9's **shell** command to fork a shell to the window. Type:

```
shell i=/wnumber & [ENTER]
```

where *number* is the number used in the **iniz** or **wcreate** command. The **i=** parameter creates an *immortal* shell. Creating an immortal shell protects the window and its shell from being destroyed if you accidentally exit the shell using [CTRL][BREAK]. If you omit the **i=** parameter, the shell is forked to the next available device window.

You now have a window that can run its own tasks. Information displayed in that window is automatically scaled to the window's size.

Opening an Overlay Window

To open an overlay window, use the Overlay Window Set function. (See OWSet in Chapter 3, "General Commands.")

Chapter 2

Overview of Commands and Parameters

The windowing commands are divided among three chapters, based on their functions.

Chapter 3 describes the *general commands*. These commands let you create windows and buffers, access buffers, set switches, and maintain the window environment.

Chapter 4 describes the *drawing commands*. Besides letting you draw all kinds of images (circles, ellipses, arcs, and boxes, to name a few), these commands also enable you to color areas or to fill them with patterns.

Chapter 5 describes the *text commands*. Use these commands to manipulate the text cursor and the text attributes. Text commands operate on hardware text screens (Screen Types 1 and 2) and graphics windows if a font is selected.

Each command description lists the command's name, code, and parameters. To call a Windowing System command using OS-9's **display** command, type `display`, followed by the command code and the values you want to supply for the parameters.

Parameters

The following is a complete list of the parameter abbreviations used in Chapters 3, 4, and 5. All parameters represent a single byte of information.

Parameter	Description
HBX	*high order byte of x value*
LBX	*low order byte of x value*
HBY	*high order byte of y value*
LBX	*low order byte of y value*
HBXo	*high order byte of x-offset value (relative)*
LBXo	*low order byte of x-offset value (relative)*
HBYo	*high order byte of y-offset value (relative)*
LBYo	*low order byte of y-offset value (relative)*
HBR	*high order byte of radius*
LBR	*low order byte of radius*

Parameter	**Description**
HBL	*high order byte of length*
LBL	*low order byte of length*
HSX	*high order byte of size in x direction*
LSX	*low order byte of size in x direction*
HSY	*high order byte of size in y direction*
LSY	*low order byte of size in y direction*
HBRx	*high order byte of radius in x direction*
LBRx	*low order byte of radius in x direction*
GRP	*GET/PUT buffer group number* (1-254)
BFN	*GET/PUT buffer number* (1-255)
LCN	*logic code number*
PRN	*palette register number* (0-15, wraps mod 15)
CTN	*color table number* (0-63, wraps mod 64)
FNM	*font number*
CPX	*character position x* (0-xmax)
CPY	*character position y* (0-ymax)
STY	*screen type*
SVS	*save switch* (0 = nosave, 1 = save area under overlay)
SZX	*size in x (columns)*
SZY	*size in y (rows)*
XDR	*dimension ratio x used with YDR as YDR/XDR*
YDR	*dimension ratio y*
BSW	*binary switch* (0 = off, 1 = on)

Chapter 3

General Commands

The general commands let you set up and customize windows. They also let you set up and access image buffers and select colors for the screen.

BColor Background Color

Function: Lets you choose a color palette register to use as the background color. See the Palette command for setting up the actual colors.

Code: 1B 33

Parameters: PRN

BoldSw Bold Switch

Function: Enables or disables boldfacing for text on graphics screens. If boldface is on, the screen displays subsequent characters in bold. If boldface is off, the screen displays subsequent characters in the regular font.

Code: 1B 3D

Parameters: BSW

```
BSW = switch
        00 = off (Default)
        01 = on
```

Notes:

- You can use BoldSw with any font.
- Boldface is not supported on hardware text screens (Screen Types 1 and 2).

Border Border Color

Function: Lets you change the palette register used for the screen border. See the Palette command for setting up the actual colors.

Code: 1B 34

Parameters: PRN

Notes:

- You set the border by selecting a palette register to use for the border register. When the actual color is changed in the palette register selected by the command, the color of the screen border changes to the new color. In general, the border register usually matches the background palette register.

CWArea Change Working Area

Function: Lets you alter the working area of the window. Normally, the system uses this call for high-level windowing, but you can use it to restrict output to a smaller area of the window.

Code: 1B 25

Parameters: CPX CPY SZX SZY

Notes:

- You cannot change a window's working area to be larger than the predefined area of the window as set by DWSet or OWSet.
- All drawing and window updating commands are done on the *current working area* of a window. The working area defaults to the entire size of the window. Scaling, when in use, is also performed relative to the current working area of a window. The CWArea command allows users to restrict the working area of a window to smaller than the full window size. Functions which might be done by opening a non-saved overlay window to draw or clear an image and then closing the overlay can be accomplished by using this command to shorten execution time where an actual overlay window is not needed.

DefColr Default Color

Function: Sets the palette registers back to their default values. The actual values of the palette registers depend on the type of monitor you are using. (See **montype** in *OS-9 Level Two Commands*.)

Code: 1B 30

Parameters: None

Notes:

- The default color definitions apply only to high-resolution graphics and text displays.
- The system sets the palette registers to a proper compatibility mode when switching to screens using the older VDG emulation modes. See the table below:

Window System Color Modes		VDG-Compatible Modes	
Palette	Color	P# Color	P# Color
00 & 08	White	00 Green	08 Black
01 & 09	Blue	01 Yellow	09 Green
02 & 10	Black	02 Blue	0A Black
03 & 11	Green	03 Red	0B Buff
04 & 12	Red	04 Buff	0C Black
05 & 13	Yellow	05 Cyan	0D Green
06 & 14	Magenta	06 Magenta	0E Black
07 & 15	Cyan	07 Orange	0F Orange

- The SetStat call lets you change the default color palette definition when using the windowing system. Default colors in the VDG-Compatible Mode cannot be changed. See the *OS-9 Level Two Technical Reference* manual for information on SetStat.
- The system's default colors are used whenever you create a new window.

DfnGPBuf Define GET/PUT Buffer

Function: Lets you define the size of the GET/PUT buffers for the system. Once you allocate a GET/PUT buffer, it remains allocated until you use the KilBuf command to delete it.

OS-9 allocates memory for GET/PUT buffers in 8K blocks that are then divided into the different GET/PUT buffers. Buffers are divided into buffer groups. Therefore, all commands dealing with GET/PUT buffers must specify both a group number and a buffer number within that group.

Code: 1B 29

Parameters: GRP BFN HBL LBL

Technical:

The buffer usage map is as follows:

Group Number	Buffer Number[1]	Use
0	1 - 255	Internal use only (returns errors)
1 - 199	1 - 255	General use by applications[2]
200 - 254	1 - 255	Reserved (Microware use only)[3]
255	1 - 255	Internal use only (returns errors)

[1] Buffer Number 0 is invalid and cannot be used.
[2] The application program should request its user ID via the GetID system call to use as its group number for buffer allocation.
[3] The standard group numbers are defined as follows:

Note: The names, buffer groups, and buffer numbers are defined in the assembly definition file. The decimal number you use to call these are in parentheses next to the name. For example, to select the Arrow pointer, Grp_Ptr and Ptr_Arr, you use 202,1 as the group/buffer number.

Grp_Fnt(200) = font group for system fonts

Fnt_S8x8(1)	= standard 8x8 font
Fnt_S6x8(2)	= standard 6x8 font
Fnt_G8x8(3)	= standard graphics font

The standard fonts are in the file SYS/StdFonts.

Grp_Clip(201) = clipboarding group (for Multiview)

Grp_Ptr(202) = graphics cursor (pointer) group

Ptr_Arr(1)	= arrow pointer (hp = 0,0)
Ptr_Pen(2)	= pencil pointer (hp = 0,0)
Ptr_LCH(3)	= large cross hair pointer (hp = 7,7)
Ptr_Slp(4)	= sleep indicator (hourglass)
Ptr_Ill(5)	= illegal indicator
Ptr_Txt(6)	= text pointer (hp = 3,3)
Ptr_SCH(7)	= small cross hair pointer (hp = 3,3)
hp	= hit point, the coordinates of the actual point on the object at which the cursor should be centered.

The standard pointers are in the file SYS/StdPtrs.

Grp_Pat2(203) = two color patterns
Grp_Pat4(204) = four color patterns
Grp_Pat6(205) = sixteen color patterns

Pat_Dot(1)	= dot pattern
Pat_Vrt(2)	= vertical line pattern
Pat_Hrz(3)	= horizontal line pattern
Pat_XHtc(4)	= cross hatch pattern
Pat_LSnt(5)	= left slanted lines
Pat_RSnt(6)	= right slanted lines
Pat_SDot(7)	= small dot pattern
Pat_BDot(8)	= large dot pattern

Each pattern is found within each of the pattern groups.

Standard patterns are in the files SYS/StdPats_2, SYS/StdPats_4, and SYS/StdPats_16.

All files have GPLoad commands imbedded in file, along with the data. To load fonts, pointers, or patterns, simply merge them to any window device: For example:

```
    merge SYS/StdFonts [ENTER]
```

sends the standard font to standard output which may be redirected to another device if the current output device is not a window device (such as when term is a VDG screen).

You only need to load fonts once for the entire system. Once a Get/Put buffer is loaded, it is available to all devices and processes in the system.

DWEnd Device Window End

Function: Ends a current device window. DWEnd closes the display window. If the window was the last device window on the screen, DWEnd also deallocates the memory used by the window. If the window is an interactive window, OS-9 automatically switches you to a new device window, if one is available.

Code: 1B 24

Parameters: None

Notes:

- DWEnd is only needed for windows that have been attached via the iniz utility or the I$Attach system call. Non-attached windows have an implied DWEnd command that is executed when you close the path.

DWProtSw Device Window Protect Switch

Function: Disables and enables device window protection. By default, device windows are protected so that you cannot overlay them with other device windows. This type of protection helps avoid the possibility of destroying the contents of either or both windows.

Code: 1B 36

Parameters: BSW

```
BSW = switch
      00 = off
      01 = on (Default)
```

Notes:

- We recommend that you not turn off device window protection. If you do, however, use extreme discretion because you might destroy the contents of the windows. OS-9 does not return an error if you request that a new window be placed over an area of the screen which is already in use by an unprotected window.

DWSet Device Window Set

Function: Lets you define a window's size and location on the physical screen. Use DWSet after opening a path to a device window.

Code: 1B 20

Parameters: STY CPX CPY SZX SZY PRN1 PRN2 PRN3

PRN1 = Foreground
PRN2 = Background
PRN3 = Border (if STY ≧ 1)

Notes:

- The **iniz** and **display** commands open paths to the device window.
- When using DWSet in a program, you must first open the device.
- Output to a new window is ignored until OS-9 receives a DWSet command, unless defaults are present in the device descriptor (/w1-/w7). If defaults are present in the device descriptors, OS-9 automatically executes DWSet, using those defaults.
- When OS-9 receives the DWSet, it allocates memory for the window, and clears the window to the current background color. If the standard font is already in memory, OS-9 assigns it as the default font. If the standard font is not in memory, you must execute a font set (Font) command after loading the fonts to produce text output on a graphics window.
- Use the Screen Type code (STY) to define the resolution and color mode of the new screen. If the screen type code is zero, OS-9 opens the window on the process's currently selected screen. If the code is 01, OS-9 opens the window on the currently displayed screen. If the code is non-zero, OS-9 allocates a new screen for the window. The following describes the acceptable screen types:

Code	Screen Size	Colors	Memory	Type
FF	Current Displayed Screen[1]			
00	Process's Current Screen			
01	40 x 24	8 & 8	2000	Text
02	80 x 24	8 & 8	4000	Text
05	640 x 192	2	16000	Graphics
06	320 x 192	4	16000	Graphics
07	640 x 192	4	32000	Graphics
08	320 x 192	16	32000	Graphics

[1] Use the Current Displayed Screen option only in procedure files to display several windows on the same physical screen. All programs should operate on that process's current screen.

- The location of the window on the physical screen is determined by the diagonal line defined by:

  (CPX,CPY) and (CPX + SZX, CPY + SZY)

- The foreground, background, and border register numbers (PRN1, PRN2, and PRN3) define the palette registers used for the foreground and background colors. See the Palette command in this chapter for more information.

- When an implicit or explicit DWSet command is done on a window, the window automatically clears to the background color.

- All windows on the screen must be of the same type (either text or graphics).

- Values in the palette register affect all windows on the screen. However, you can choose which register to use for foreground and background for each window. That is, OS-9 maintains palette registers and border register numbers for the entire screen and foreground and background register numbers for each individual window.

- OS-9 deallocates memory for a screen when you terminate the last window on that screen.

FColor Foreground Color

Function: Lets you select a color palette register for the foreground color. See the Palette command for setting the actual colors.

Code: 1B 32

Parameters: PRN

Font Select Font

Function: Lets you select/change the current font. Before you can use this command, the font must be loaded into the specified GET/PUT group and buffer (using GPLoad). See the GPLoad command for information on loading font buffers.

Code: 1B 3A

Parameters: GRP BFN

Notes:

- You can select proportional spacing for the font by using PropSw.
- All font data is a 2-color bit map of the font.
- Each character in the font data consists of 8 bytes of data. The first byte defines the top scan line, the second byte defines the second scan line, and so on. The high-order bit of each byte defines the first pixel of the scan line, the next bit defines the next pixel, and so on. For example, the letter "A" would be represented like this:

```
Byte   Pixel Representation
 10    . . . # . . . .
 28    . . # . # . . .
 44    . # . . . # . .
 44    . # . . . # . .
 7c    . # # # # # . .
 44    . # . . . # . .
 44    . # . . . # . .
 00    . . . . . . . .
```

  Note that 6x8 fonts ignore the last 2 bits per byte.

- The fonts are ordered with characters in the following ranges:

$00-$1F	International characters (see mapping below)
$20-$7F	Standard ASCII characters

International characters or any characters in the font below character $20 (hex) are printed according to the following table:

Character position in font	Char1	or	Char2
$00	$C1		$E1
$01	$C2		$E2
$02	$C3		$E3
$03	$C4		$E4
$04	$C5		$E5
$05	$C6		$E6
$06	$C7		$E7
$07	$C8		$E8
$08	$C9		$E9
$09	$CA		$EA
$0A	$CB		$EB
$0B	$CC		$EC
$0C	$CD		$ED
$0D	$CE		$EE
$0E	$CF		$EF
$0F	$D0		$F0
$10	$D1		$F1
$11	$D2		$F2
$12	$D3		$F3
$13	$D4		$F4
$14	$D5		$F5
$15	$D6		$F6
$16	$D7		$F7
$17	$D8		$F8
$18	$D9		$F9
$19	$DA		$FA
$1A	$AA		$BA
$1B	$AB		$BB
$1C	$AC		$BC
$1D	$AD		$BD
$1E	$AE		$BE
$1F	$AF		$BF

GCSet Graphics Cursor Set

Function: Creates a GET/PUT buffer for defining the graphics cursor that the system displays. You must use GCSet to display a graphic cursor.

Code: 1B 39

Parameters: GRP BFN

Notes:

- To turn off the graphics cursor specify GRP as 00.
- A system standard buffer or a user-defined buffer can be used for the graphics cursor.

GetBlk Get Block

Function: Saves an area of the screen to a GET/PUT buffer. Once the block is saved, you can put it back in its original location or in another on the screen, using the PutBlk command.

Code: 1B 2C

Parameters: GRP BFN HBX LBX HBY LBY HSX LSX HSY LSY

HBX/LBX	=	*x-location of block* (upper left corner)
HBY/LBX	=	*y-location of block*
HSX/LSX	=	*x-dimension of block*
HSY/LSY	=	*y-dimension of block*

Notes:

- The GET/PUT buffer maintains information on the size of the block stored in the buffer so that the PutBlk command works more automatically.
- If the GET/PUT buffer is not already defined, GetBlk creates it. If the buffer is defined, the data must be equal to or smaller than the original size of the buffer.

GPLoad GET/PUT Buffer Load

Function: Preloads GET/PUT buffers with images that you can move to the screen later, using PutBlk.

If the GET/PUT buffer is not already created, GPLoad creates it.

If the buffer was previously created, the size of the passed data must be equal to or smaller than the original size of the buffer. Otherwise, GPLoad truncates the data to the size of the buffer.

Code: 1B 2B

Parameters: GRP BFN STY HSX LSX HSY LSY HBL LBL (Data...)

STY = *format*
HSX/LSX = *x-dimension of stored block*
HSY/LSY = *y-dimension of stored block*
HBL/LBL = *number of bytes in data*

Notes:

- Buffers are maintained in a linked list system.
- Buffers to be used most should be allocated last to minimize the search time in finding the buffers.
- When loading a Font GET/PUT Buffer, the parameters are as follows:

  GRP BFN STY HSX LSX HSY LSY HBL LBL (Data...)

  GRP = 254
  STY = 5
  HSX/LSX = *x-dimension size of Font 6 or 8*
  HSY/LSY = *y-dimension size of Font 8*
  HBL/LBL = *size of font data* (not including this header information)

See the Font command for more information on font data.

KilBuf Kill GET/PUT Buffer

Function: Deallocates the buffer specified by the group and buffer number. To deallocate the entire group of buffers, set the buffer number to 0.

Code: 1B 2A

Parameters: GRP BFN

Notes:

- KilBuf returns memory used by the buffer to a free list. When an entire block of memory has been put on the free list, the block is returned to the system.

LSet Logic Set

Function: Lets you create special effects by specifying the type of logic used when storing data, which represents an image, to memory. The specified logic code is used by all draw commands until you either choose a new logic or turn off the logic operation. To turn off the logic function, set the logic code to 00.

Code: 1B 2F

Parameters: LCN

LCN = *logic code number*

00	=	No logic code; store new data on screen
01	=	AND new data with data on screen
02	=	OR new data with data on screen
03	=	XOR new data with data on screen

Notes:

- The following tables summarize logic operations in bit manipulations:

AND	First Operand	Second Operand	Result
	1	1	1
	1	0	0
	0	1	0
	0	0	0

OR	First Operand	Second Operand	Result
	1	1	1
	1	0	1
	0	1	1
	0	0	0

XOR	First Operand	Second Operand	Result
	1	1	0
	1	0	1
	0	1	1
	0	0	0

- Data items are represented as palette register numbers in memory. Since logic is performed on the palette register number and not the colors in the registers, you should choose colors for palette registers carefully so that you obtain the desired results. You may want to choose the colors for the palette registers so that LSet appears to *and*, *or*, and *xor* the colors rather than the register numbers. For example:

Palette #	Color	Alternative Order
0	White	Black
1	Blue	Blue
2	Black	Green
3	Green	White

OWEnd Overlay Window End

Function: Ends a current overlay window. OWEnd closes the overlay window and deallocates memory used by the window. If you opened the window with a *save switch* value of hexadecimal 01, OS-9 restores the area under the window. If you did not, OS-9 does not restore the area and any further output is sent to the next lower overlay window or to the device window, if no overlay window exists.

Code: 1B 23

Parameters: None

OWSet Overlay Window Set

Function: Use OWSet to create an overlay window on an existing device window. OS-9 reconfigures current device window paths to use a new area of the screen as the current logical device window.

Code: 1B 22

Parameters: SVS CPX CPY SZX SZY PRN1 PRN2

SVS = *save switch*
- 00 = Do not save area overlayed
- 01 = Save area overlayed and restore at close

PRN1 = *background palette register*
PRN2 = *foreground palette register*

Notes:

- If you set SVS to zero, any writes to the new overlay window destroy the area under the window. You might want to set SVS to zero if your system is already using most of its available memory. You might also set SVS to zero whenever it takes relatively little time to redraw the area under the overlay window once it is closed.
- If you have ample memory, specify SVS as 1. Doing this causes the system to save the area under the new overlay window. The system restores the area when you terminate the new overlay window. (See OWEnd.)
- The size of the overlay window is specified in standard characters. Use the same resolution (number of characters) as the device window that will reside beneath the overlay window. Have your program determine the original size of the device window at startup (using the SS.ScSiz GETSTAT call), if the device window does not cover the entire screen. See the *OS-9 Level Two Technical Reference* manual for information on the SS.ScSiz GETSTAT call.

- Overlay windows can be created on top of other overlay windows; however, you can only write to the top most window. Overlay windows are "stacked" on top of each other logically. To get back down to a given overlay, you must close (OWEnd) any overlay windows that reside on top of the desired overlay window.
- Stacked overlay windows do not need to reside directly on top of underlying overlay windows. However, all overlay windows must reside within the boundaries of the underlying device window.

Palette Change Palette

Function: Lets you change the color associated with each of the 16 palette registers.

Code: 1B 31

Parameters: PRN CTN

Notes:

- Changing a palette register value causes all areas of the screen using that palette register to change to the new color. In addition, if the border is set to that palette register, the border color also changes. See the Border command for more information.

- Colors are made up by setting the red, green, and blue bits in the color byte which is inserted in the palette register. The bits are laid out as follows:

Bit	Color
0	Blue low
1	Green low
2	Red low
3	Blue high
4	Green high
5	Red high
6	unused
7	unused

By using six bits for color (2 each for red, green and blue) there is a possibility of 64 from which to choose. Some of the colors are defined as shown:

White	:	00111111	=	$3F (all color bits set)
Black	:	00000000	=	$00 (no color bits set)
Standard Blue	:	00001001	=	$09 (both blue bits set)
Standard Green	:	00010010	=	$12 (both green bits set)
Standard Red	:	00100100	=	$24 (both red bits set)

Note: These colors are for RGB monitors. The composite monitors use a different color coding and do not directly match pure RGB colors. To get composite color from the RGB colors, the system uses conversion tables. The colors were assigned to match the RGB colors as close as possible. There are, however, a wider range of composite colors, so the colors without direct matches were assigned to the closest possible match. The white, black, standard green, and standard orange are the same in both RGB and composite.

PropSw Proportional Switch

Function: Enables and disables the automatic proportional spacing of characters. Normally, characters are not proportionally spaced.

Code: 1B 3F

Parameters: BSW

```
BSW = switch
      00 = off (Default)
      01 = on
```

Notes:

- Any standard software font used in a graphics screen can be proportionally spaced.
- Proportional spacing is not supported on hardware text screens.

PSet Pattern Set

Function: Selects a preloaded GET/PUT buffer as a pattern RAM array. This pattern is used with all draw commands until you either change the pattern or turn it off by passing a parameter of 00 as GRP (Group Number).

Code: 1B 2E

Parameters: GRP BFN

Notes:

- The pattern array is a 32 x 8 pixel representation of graphics memory. The color mode defines the number of bits per pixel and pixels per byte. So, be sure to take the current color mode into consideration when creating a pattern array.
- The GET/PUT buffer can be of any size, but only the number of bytes as described by the following table are used:

Color Mode	Size of Pattern Array
2	4 bytes x 8 = 32 bytes (1 bit per pixel)
4	8 bytes x 8 = 64 bytes (2 bits per pixel)
16	16 bytes x 8 = 128 bytes (4 bits per pixel)

- The buffer must contain at least the number of bytes required by the current color mode. If the buffer is larger than required, the extra bytes are ignored.
- To turn off patterning, set GRP to 00.

- The following example creates a two color pattern of vertical lines. A two color pattern is made up of 1's and 0's. The diagram below shows the bit set pattern (note that one pixel is equal to one bit):

  ```
  10101010101010101010101010101010
  10101010101010101010101010101010
  10101010101010101010101010101010
  10101010101010101010101010101010
  10101010101010101010101010101010
  10101010101010101010101010101010
  10101010101010101010101010101010
  10101010101010101010101010101010
  ```

  When the binary for the 2x8 pixel data is compressed into byte data, notice that each row consists of 4 bytes of data. The pattern now looks like this:

  ```
  $55 $55 $55 $55
  $55 $55 $55 $55
  $55 $55 $55 $55
  $55 $55 $55 $55                $55 = 01010101
  $55 $55 $55 $55
  $55 $55 $55 $55
  $55 $55 $55 $55
  $55 $55 $55 $55
  ```

  To load the pattern in the system, use the GPLoad command. To load this particular pattern into Group 2 and Buffer 1, the command would be:

  ```
  display 1b 2b 02 01 00 20 00 08 00 20 55 55 ...55 [ENTER]
             |  |  |     |     |     |  |________|
             |  |  |     |     |     |   32 times
             |  |  |     |     |     number of bytes (32)
             |  |  |     |     y size of pattern (8)
             |  |  |     x size of pattern (32)
             |  |  buffer number
             |  group number
             GPLoad code
  ```

- When making a pattern using 4 colors, a pixel is made up of two bits instead of one. This means that the pattern consists of 64 bytes instead of 32. The diagram below shows the bit set pattern for the same vertical pattern using 4 colors:

  ```
  1100110011001100110011001100110011001100110011001100110011001100
  1100110011001100110011001100110011001100110011001100110011001100
  1100110011001100110011001100110011001100110011001100110011001100
  1100110011001100110011001100110011001100110011001100110011001100
  1100110011001100110011001100110011001100110011001100110011001100
  1100110011001100110011001100110011001100110011001100110011001100
  1100110011001100110011001100110011001100110011001100110011001100
  1100110011001100110011001100110011001100110011001100110011001100
  ```

  When the binary for the 4x8 pixel data is compressed into byte data, notice that each row consists of 8 bytes of data. The pattern now looks like this:

  ```
  $CC $CC $CC $CC $CC $CC $CC $CC
  $CC $CC $CC $CC $CC $CC $CC $CC
  $CC $CC $CC $CC $CC $CC $CC $CC
  $CC $CC $CC $CC $CC $CC $CC $CC        $CC=1100
  $CC $CC $CC $CC $CC $CC $CC $CC
  $CC $CC $CC $CC $CC $CC $CC $CC
  $CC $CC $CC $CC $CC $CC $CC $CC
  $CC $CC $CC $CC $CC $CC $CC $CC
  ```

  To load the pattern in the system, use the GPLoad command as described for the 2 color example but specify $40 (64) bytes instead of $20 (32).

- When making a pattern using 16 colors, a pixel is made up of four bits instead of one. This means that the pattern consists of 128 bytes. Each line in the bit pattern would look like this:

  11110000...{repeat pattern for 16 total sets}...11110000

  When the binary for the 8x8 pixel data is compressed into byte data, the pattern is a series of $F0.

  To load the pattern in the system, use the GPLoad command and specify $80 (128) bytes as the size.

PutBlk Put Block

Function: Moves a GET/PUT buffer, previously copied from the screen or loaded with GPLoad, to an area of the screen.

Code: 1B 2D

Parameters: GRP BFN HBX LBX HBY LBY

HBX/LBX = *x-location of block* (upper left corner)
HBY/LBY = *y-location of block*

Notes:

- The dimensions of the block were saved in the GET/PUT buffer when you created it. OS-9 uses these dimensions when restoring the buffer.
- The screen type conversion is automatically handled by the PutBlk routine in the driver.
- GET/PUT buffers cannot be scaled. The image will be clipped if it does not fit within the window.

ScaleSw **Scale Switch**

Function: Disables and enables automatic scaling. Normally, automatic scaling is enabled. When scaling is enabled, coordinates refer to a relative location in a window that is proportionate to the size of the window. When scaling is disabled, coordinates passed to a command will be the actual coordinates for that type of screen relative to the origin of the window.

Code: 1B 35

Parameters: BSW

BSW = *switch*
00 = off
01 = on (Default)

Notes:

- A useful application of disabled scaling is the arrangement of references between a figure and text.
- All coordinates are relative to the window's origin (0,0). The valid range for the coordinates:

  Scaling enabled:

  y = 0-191
  x = 0-639

  Scaling disabled:

  y = 0-*size of y - 1*
  x = 0-*size of x - 1*

Select Window Select

Function: Causes the current process's window to become the active (display) window. You can select a different window by using the form:

```
display 1B 21 >/wnumber
```

where *number* is the desired window number. If the process that executes the select is running on the current interactive (input/display) window, the selected window becomes the interactive window, and the other window becomes passive.

Code: 1B 21

Parameters: None

Notes:

- The keyboard is attached to the process's selected window through the use of the CLEAR key. This lets you input data from the keyboard to different windows by using the CLEAR key to select the window.
- All display windows that occupy the same screen are also displayed.
- The device window which owns the keyboard is the current interactive window. The interactive window is always the window being displayed. Only one process may receive input from the keyboard at a time. Many processes may be changing the output information on their own windows; however, you can only see the information that is displayed on the interactive window and any other window on the same screen as the interactive window.

TCharSw Transparent Character Switch

Function: Defines the character mode to be used when putting characters on the graphics screens.

In the default mode (transparent off), the system uses block characters that draw the entire foreground and background for that cell.

When in transparent mode, the only pixels that are changed are the ones where the character actually has pixels set in its font. When transparent mode is off, all pixels in the character block are set: foreground or font pixels in the foreground color and others in the background color.

Code: 1B 3C

Parameters: BSW

BSW = *switch*
- 00 = off (Default)
- 01 = on

Chapter 4

Drawing Commands

All drawing commands relate to an invisible point of reference on the screen called the *draw pointer*. Originally, the draw pointer is at position 0,0. You can change the position by using the SetDPtr and RSetDPtr commands described in this chapter. In addition, some draw commands automatically update the draw pointer. For example, the LineM command draws a line from the current draw pointer position to the specified end coordinates and moves the draw pointer to those end coordinates. The Line command draws a line but does not move the pointer.

Also, note that all draw commands are affected by the pattern and logic commands described in Chapter 3.

Do not confuse the draw pointer with the graphics cursor. The graphics cursor is the graphic representation of the mouse/joystick position on the screen.

In this chapter, commands that use relative coordinates (*offsets*) are listed with their counterparts that use absolute coordinates. For example, RSetDPtr is listed under SetDPtr.

Arc3P Draw Arc

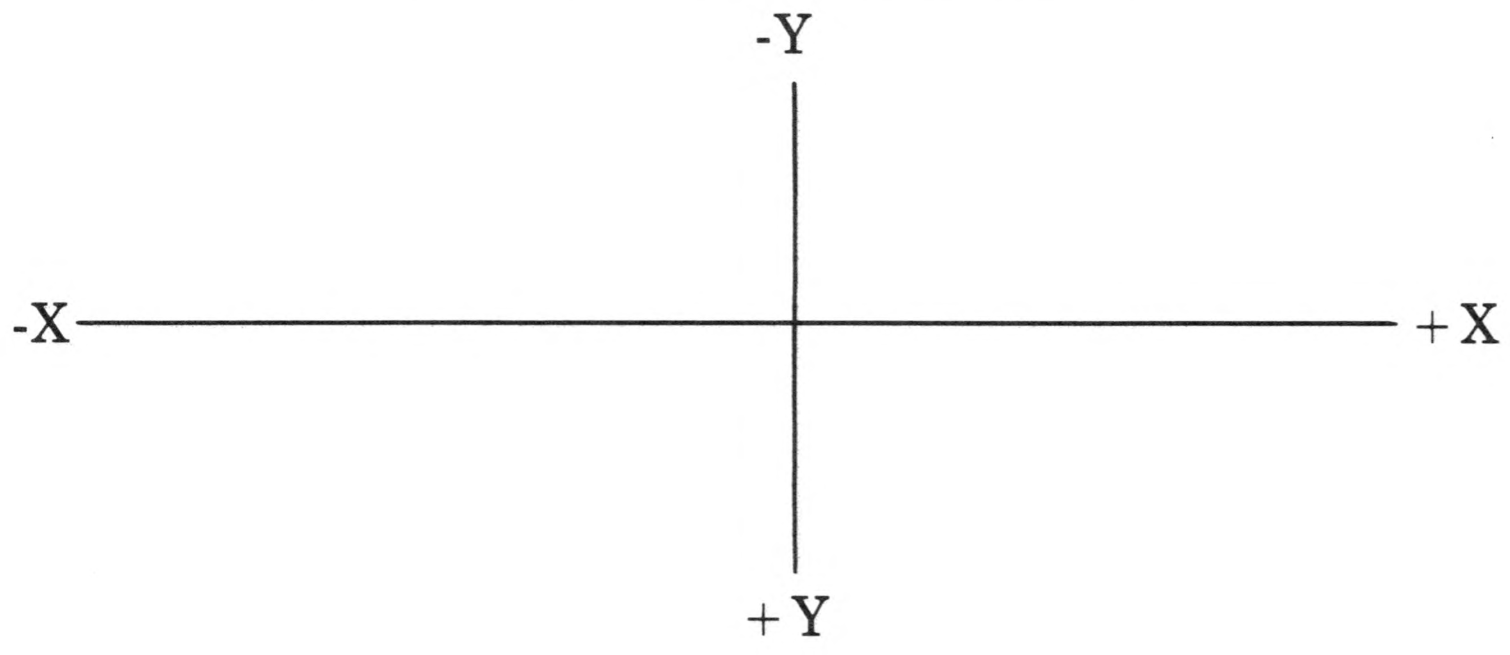
Function: Draws an arc with its midpoint at the current draw pointer position. You specify the curve by both the X and Y dimensions, as you do an ellipse. In this way, you can draw either elliptical or circular arcs. The arc is clipped by a line defined by the (X01,Y01) - (X02,Y02) coordinates. These coordinates are signed 16 bit values and are relative to the center of the ellipse. The draw pointer remains in its original position.

Code: 1B 52

Parameters: HBRx LBRx HBRy LBRy HX01 LX01 HY01 LY01 HX02 LX02 HY02 LY02

Notes:

- The resulting arc depends on the order in which you specify the line coordinates. Arc3P first draws the line from Point 1 to Point 2 and then draws the ellipse in a clockwise direction.
- The coordinates of the screen are as follows:



Bar Draw Bar

Function: Draws and fills a rectangle that is defined by the diagonal line from the current draw pointer position to the specified position. The box is drawn in the current foreground color. The draw pointer returns to its original location.

Code: 1B 4A

Parameters: HBX LBX HBY LBY

RBar Relative Draw Bar

Function: Draws and fills a rectangle that is defined by the diagonal line from the current draw pointer position to the point specified by the offsets. The box is drawn in the current foreground color. The draw pointer returns to its original location. This is a relative command.

Code: 1B 4B

Parameters: HBXo LBXo HBYo LBYo

Box Draw Box

Function: Draws a rectangle that is defined by the diagonal line from the current draw pointer position to the specified position. The box is drawn in the current foreground color. The draw pointer returns to its original location.

Code: 1B 48

Parameters: HBX LBX HBY LBY

RBox Relative Draw Box

Function: Draws a rectangle that is defined by the diagonal line from the current draw pointer position to the point specified by the offsets. The box is drawn in the current foreground color. The draw pointer returns to its original location. This is a relative command.

Code: 1B 49

Parameters: HBXo LBXo HBYo LBYo

Circle Draw Circle

Function: Draws a circle of the specified radius with the center of the circle at the current draw pointer position. The circle is drawn in the current foreground color. The draw pointer remains in its original location.

Code: 1B 50

Parameters: HBR LBR

Ellipse **Draw Ellipse**

Function: Draws an ellipse with its center at the current draw pointer position. The X value specifies the horizontal radius, and the Y value specifies the vertical radius. The ellipse is drawn in the current foreground color. The draw pointer remains in its original location. This is a relative command.

Code: 1B 51

Parameters: HBRx LBRx HBRy LBRy

FFill Flood Fill

Function: Fills the area where the background is the same color as the draw pointer. Filling starts at the current draw pointer position, using the current foreground color. The draw pointer returns to its original location. This is a relative command.

Code: 1B 4F

Parameters: None

Line Draw Line

Function: Draws a line from the current draw pointer position to the specified point, using the current foreground color. The draw pointer returns to its original location.

Code: 1B 44

Parameters: HBX LBX HBY LBY

RLine Relative Draw Line

Function: Draws a line from the current draw pointer position to the point specified by the *x,y* offsets, using the current foreground color. The draw pointer returns to its original location. This is a relative command.

Code: 1B 45

Parameters: HBXo LBXo HBYo LBYo

LineM Draw Line and Move

Function: Draws a line from the current draw pointer position to the specified point, using the current foreground color. The draw pointer stays at the new location.

Code: 1B 46

Parameters: HBX LBX HBY LBY

RLineM Relative Draw Line and Move

Function: Draws a line from the current draw pointer position to the point specified by the offsets, using the current foreground color. The draw pointer stays at the new location. This is a relative command.

Code: 1B 47

Parameters: HBXo LBXo HBYo LBYo

Point Draw Point

Function: Draws a pixel at the specified coordinates, using the current foreground color.

Code: 1B 42

Parameters: HBX LBX HBY LBY

RPoint Relative Draw Point

Function: Draws a pixel at the location specified by the offsets, using the current foreground color. This is a relative command.

Code: 1B 43

Parameters: HBXo LBXo HBYo LBYo

PutGC Put Graphics Cursor

Function: Puts and displays the graphics cursor at the specified location. The coordinates passed to this command are not window-relative. The horizontal range is 0 to 639. The vertical range is 0 to 191. The default position is 0,0.

This command is useful for applications running under GrfInt so that you can display a graphics cursor under WindInt even if you don't want mouse control of the cursor.

Code: 1B 4E

Parameters: HBX LBX HBY LBY

SetDPtr Set Draw Pointer

Function: Sets the draw pointer to the specified coordinates. The new draw pointer position is used as the beginning point in the next draw command if other coordinates are not specified.

Code: 1B 40

Parameters: HBX LBX HBY LBY

RSetDPtr Relative Set Draw Pointer

Function: Sets the draw pointer to the point specified by the offsets. The new draw pointer position is used as the beginning point in the next draw command if other coordinates are not specified. This is a relative command.

Code: 1B 41

Parameters: HBXo LBXo HBYo LBYo

Chapter 5

Text Commands

The text commands let you control the cursor's position and movement and also the way text prints on the display. These commands can be used on either text or graphics windows.

The text commands are:

Code	**Description**
01	Homes the cursor.
02 *x y*	Positions cursor to *x*,*y*. Specify coordinates as ($x+\$20$) and ($y+\$20$).
03	Erases the current line.
04	Erases from the current character to the end of the line.
05 20	Turns off the cursor.
05 21	Turns on the cursor.
06	Moves the cursor right one character.
07	Rings the bell.
08	Moves the cursor left one character.
09	Moves the cursor up one line.
0A	Moves the cursor down one line.
0B	Erases from the current character to the end of the screen.
0C	Erases the entire screen and homes the cursor.
0D	Sends a carriage return.
1F 20	Turns on reverse video.
1F 21	Turns off reverse video.
1F 22	Turns on underlining.
1F 23	Turns off underlining.
1F 24	Turns on blinking.[1]

Code	Description
1F 25	Turns off blinking.[1]
1F 30	Inserts a line at the current cursor position.
1F 31	Deletes the current line.
1B 3C BSW	See TCharSw in Chapter 3.[2]
1B 3D BSW	See BoldSw in Chapter 3.[2]
1B 3F BSW	See PropSw in Chapter 3.[2]

[1] Blink is not supported for text on graphics screens.

[2] These characteristics are supported for text on graphics screens only.

Index